

EXHIBIT Q

[Sign in](#)

[android](#) / [platform](#) / [tools](#) / [base](#) / [refs/heads/mirror-goog-studio-main](#) / [.](#) / [lint](#) / [libs](#) / [lint-checks](#) / [src](#) / [main](#) / [java](#) / [com](#) / [android](#) / [tools](#) / [lint](#) / [checks](#) / [WakelockDetector.java](#)

blob: 7cd4fba9c9abe57f7b85c7ed788dba2b9eb963af [file] [log] [blame]			
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
1 /* 2 * Copyright (C) 2012 The Android Open Source Project 3 * 4 * Licensed under the Apache License, Version 2.0 (the "License"); 5 * you may not use this file except in compliance with the License. 6 * You may obtain a copy of the License at 7 * 8 * http://www.apache.org/licenses/LICENSE-2.0 9 * 10 * Unless required by applicable law or agreed to in writing, software 11 * distributed under the License is distributed on an "AS IS" BASIS, 12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. 13 * See the License for the specific language governing permissions and 14 * limitations under the License. 15 */ 16 17 package com.android.tools.lint.checks;			
Tor Norbye	0e717c7	2013-01-02 10:32:38 -0800	[diff] [blame]
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
18 import com.android.annotations.NonNull; 19 import com.android.annotations.Nullable; 20 import com.android.tools.lint.checks.ControlFlowGraph.Node; 21 import com.android.tools.lint.detector.api.Category; 22 import com.android.tools.lint.detector.api.ClassContext; 23 import com.android.tools.lint.detector.api.ClassScanner; 24 import com.android.tools.lint.detector.api.Context; 25 import com.android.tools.lint.detector.api.Detector; 26 import com.android.tools.lint.detector.api.Implementation; 27 import com.android.tools.lint.detector.api.Issue; 28 import com.android.tools.lint.detector.api.JavaContext; 29 import com.android.tools.lint.detector.api.LintFix; 30 import com.android.tools.lint.detector.api.LintFixFix; 31 import com.android.tools.lint.detector.api.Location; 32 import com.android.tools.lint.detector.api.Scope; 33 import com.android.tools.lint.detector.api.Severity; 34 import com.android.tools.lint.detector.api.SourceCodeScanner;			
Tor Norbye	d9210bd	2018-01-05 21:17:11 +0100	[diff] [blame]
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
Tor Norbye	c03df13	2013-01-17 13:26:19 -0800	[diff] [blame]
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	[diff] [blame]
Tor Norbye	78257e6	2018-04-18 17:26:30 -0700	[diff] [blame]
Tor Norbye	a44cd04	2017-04-14 21:19:23 -0700	[diff] [blame]
Tor Norbye	0dedf90	2016-12-20 10:19:58 +0000	[diff] [blame]
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
Tor Norbye	d9210bd	2018-01-05 21:17:11 +0100	[diff] [blame]
Matthew Gharrity	32788cf	2020-07-18 14:14:40 -0700	[diff] [blame]
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	[diff] [blame]
Tor Norbye	9719863	2016-10-24 08:39:14 -0700	[diff] [blame]
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	[diff] [blame]
Tor Norbye	9719863	2016-10-24 08:39:14 -0700	[diff] [blame]
Tor Norbye	0dedf90	2016-12-20 10:19:58 +0000	[diff] [blame]
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
35 import com.intelli.j.psi.PsiClass; 36 import com.intelli.j.psi.PsiMethod; 37 import java.util.Arrays; 38 import java.util.Collections; 39 import java.util.List; 40 import org.jetbrains.uast.UCallExpression; 41 import org.objectweb.asm.Opcodes; 42 import org.objectweb.asm.tree.AbstractInsnNode; 43 import org.objectweb.asm.tree.ClassNode; 44 import org.objectweb.asm.tree.InsnList; 45 import org.objectweb.asm.tree.JumpInsnNode; 46 import org.objectweb.asm.tree.LdcInsnNode; 47 import org.objectweb.asm.tree.MethodInsnNode; 48 import org.objectweb.asm.tree.MethodNode; 49 import org.objectweb.asm.tree.analysis.AnalyzerException; 50 51			
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
Tor Norbye	ff37cc8	2018-03-09 07:56:23 -0800	[diff] [blame]
52 /** 53 * Checks for problems with wakelocks (such as failing to release them) which can lead to 54 * unnecessary battery usage. 55 */ 56 public class WakelockDetector extends Detector implements ClassScanner, SourceCodeScanner { 57 public static final String ANDROID_APP_ACTIVITY = "android.app.Activity"; 58 59 /** Problems using wakelocks */ 60 public static final Issue ISSUE = 61 Issue.create(62 "Wakelock", 63 "Incorrect 'Wakelock' usage", 64 "Failing to release a wakelock properly can keep the Android device in " 65 + "a high power mode, which reduces battery life. There are several causes " 66 + "of this, such as releasing the wake lock in 'onDestroy()' instead of in " 67 + "'onPause()', failing to call 'release()' in all possible code paths after " 68 + "an 'acquire()', and so on.\n", 69 + "\n", 70 + "NOTE: If you are using the lock just to keep the screen on, you should " 71 + "strongly consider using 'FLAG_KEEP_SCREEN_ON' instead. This window flag " 72 + "will be correctly managed by the platform as the user moves between " 73 + "applications and doesn't require a special permission. See " 74 + "https://developer.android.com/reference/android/view/WindowManager.LayoutParams.html#FLAG_KEEP_SCREEN_ON.", 75 Category.PERFORMANCE, 76 9, 77 Severity.WARNING, 78 new Implementation(WakelockDetector.class, Scope.CLASS_FILE_SCOPE)) 79 .setAndroidSpecific(true);			
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	[diff] [blame]
Tor Norbye	ff37cc8	2018-03-09 07:56:23 -0800	[diff] [blame]
80 81 /** Using non-timeout version of wakelock acquire */ 82 public static final Issue TIMEOUT = 83 Issue.create(84 "WakelockTimeout", 85 "Using wakelock without timeout", 86 "Wakelocks have two acquire methods: one with a timeout, and one without. " 87 + "You should generally always use the one with a timeout. A typical " 88 + "timeout is 10 minutes. If the task takes longer than it is critical " 89 + "that it happens (i.e. can't use 'JobScheduler') then maybe they " 90 + "should consider a foreground service instead (which is a stronger " 91 + "run guarantee and lets the user know something long/important is " 92 + "happening).", 93 Category.PERFORMANCE, 94 9, 95 Severity.WARNING, 96 new Implementation(WakelockDetector.class, Scope.JAVA_FILE_SCOPE)) 97 .setAndroidSpecific(true);			
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	[diff] [blame]
Tor Norbye	9719863	2016-10-24 08:39:14 -0700	[diff] [blame]
98 private static final String WAKELOCK_OWNER = "android/os/PowerManager\$WakeLock"; 99 private static final String RELEASE_METHOD = "release"; 100 private static final String ACQUIRE_METHOD = "acquire"; 101 private static final String IS_HELD_METHOD = "isHeld"; 102 private static final String POWER_MANAGER = "android/os/PowerManager"; 103 private static final String NEW_WAKE_LOCK_METHOD = "newWakeLock"; 104 105			
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
Tor Norbye	ff37cc8	2018-03-09 07:56:23 -0800	[diff] [blame]
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	[diff] [blame]
106 /** Constructs a new {@link WakelockDetector} */ 107 public WakelockDetector() {} 108 109 @Override 110 public void afterCheckRootProject(@NonNull Context context) {			

```
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 111         if (mHasAcquire && !mHasRelease && context.getDriver().getPhase() == 1) {
112             // Gather positions of the acquire calls
113             context.getDriver().requestRepeat(this, Scope.CLASS_FILE_SCOPE);
114         }
115     }
116
117     // ---- Implements ClassScanner ----
118
119     /** Whether any {@code acquire()} calls have been encountered */
120     private boolean mHasAcquire;
121
122     /** Whether any {@code release()} calls have been encountered */
123     private boolean mHasRelease;
124
125     @Override
126     @Nullable
127     public List<String> getApplicableCallNames() {
128         return Arrays.asList(ACQUIRE_METHOD, RELEASE_METHOD, NEW_WAKE_LOCK_METHOD);
129     }
130
131     @Override
132     public void checkCall(
133         @NonNull ClassContext context,
134         @NonNull ClassNode classNode,
135         @NonNull MethodNode method,
136         @NonNull MethodInsnNode call) {
137         if (!context.getProject().getReportIssues()) {
138             // If this is a library project not being analyzed, ignore it
139             return;
140         }
141
142         if (call.owner.equals(WAKELOCK_OWNER)) {
143             String name = call.name;
144             if (name.equals(ACQUIRE_METHOD)) {
145                 if (call.desc.equals(
146                     "(J)V") { // acquire(long timeout) does not require a corresponding release
147                     return;
148                 }
149                 mHasAcquire = true;
150
151                 if (context.getDriver().getPhase() == 2) {
152                     assert !mHasRelease;
153                     context.report(
154                         ISSUE,
155                         method,
156                         call,
157                         context.getLocation(call),
158                         "Found a wakelock 'acquire()' but no 'release()' calls anywhere");
159                 } else if (context.getDriver().getPhase() == 1;
160                     // Perform flow analysis in this method to see if we're
161                     // performing an acquire/release block, where there are code paths
162                     // between the acquire and release which can result in the
163                     // release call not getting reached.
164                     checkFlow(context, classNode, method, call);
165                 }
166             } else if (name.equals(RELEASE_METHOD)) {
167                 mHasRelease = true;
168             }
169
170             // See if the release is happening in an onDestroy method, in an activity.
171             if ("onDestroy".equals(method.name)) {
172                 PsiClass psiClass = context.findPsiClass(classNode);
173                 PsiClass activityClass = context.findPsiClass(ANDROID_APP_ACTIVITY);
174                 if (psiClass != null
175                     && activityClass != null
176                     && psiClass.isInheritor(activityClass, true)) {
177                     context.report(
178                         ISSUE,
179                         method,
180                         call,
181                         context.getLocation(call),
182                         "Wakelocks should be released in 'onPause', not 'onDestroy'");
183                 }
184             }
185         } else if (call.owner.equals(POWER_MANAGER)) {
186             if (call.name.equals(NEW_WAKE_LOCK_METHOD)) {
187                 AbstractInsnNode prev = Lint.getPrevInstruction(call);
188                 if (prev == null) {
189                     return;
190                 }
191                 prev = Lint.getPrevInstruction(prev);
192                 if (prev == null || prev.getOpcode() != Opcodes.LDC) {
193                     return;
194                 }
195                 LdcInsnNode ldc = (LdcInsnNode) prev;
196                 Object constant = ldc.cst;
197                 if (constant instanceof Integer) {
198                     int flag = (Integer) constant;
199                     // Constant values are copied into the bytecode so we have to compare
200                     // values; however, that means the values are part of the API
201                     final int PARTIAL_WAKE_LOCK = 0x00000001;
202                     final int ACQUIRE_CAUSES_WAKEUP = 0x10000000;
203                     final int both = PARTIAL_WAKE_LOCK | ACQUIRE_CAUSES_WAKEUP;
204                     if ((flag & both) == both) {
205                         context.report(
206                             ISSUE,
207                             method,
208                             call,
209                             context.getLocation(call),
210                             "Should not set both 'PARTIAL_WAKE_LOCK' and 'ACQUIRE_CAUSES_WAKEUP'. "
211                                 + "If you do not want the screen to turn on, get rid of "
212                                 + "'ACQUIRE_CAUSES_WAKEUP'");
213                     }
214                 }
215             }
216         }
217     }
218
219     private static void checkFlow(
220         @NonNull ClassContext context,
221         @NonNull ClassNode classNode,
222         @NonNull MethodNode method,
223         @NonNull MethodInsnNode acquire) {
224         final InsnList instructions = method.instructions;
225         MethodInsnNode release = null;
226
227         // Find release call
228         for (int i = 0, n = instructions.size(); i < n; i++) {
229             AbstractInsnNode instruction = instructions.get(i);
230             int type = instruction.getType();
231             if (type == AbstractInsnNode.METHOD_INSN) {
232                 MethodInsnNode call = (MethodInsnNode) instruction;
```

```
Tor Norbye ff37cc8 2018-03-09 07:56:23 -0800 (diff) (blame) 234         if (call.name.equals(RELEASE_METHOD) && call.owner.equals(WAKELOCK_OWNER)) {
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 235             release = call;
236             break;
237         }
238     }
239 }
240
241 if (release == null) {
242     // Didn't find both acquire and release in this method; no point in doing
243     // local flow analysis
244     return;
245 }
246
247 try {
248     MyGraph graph = new MyGraph();
249     ControlFlowGraph.create(graph, classNode, method);
250
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 251     int status = dfs(graph.getNode(acquire));
252     if ((status & SEEN_RETURN) != 0) {
253         String message;
254         if ((status & SEEN_EXCEPTION) != 0) {
255             message = "The 'release()' call is not always reached (via exceptional flow)";
Tor Norbye 8a365ef 2014-09-14 12:26:36 -0700 (diff) (blame) 256         } else {
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 257             message = "The 'release()' call is not always reached";
Tor Norbye 8a365ef 2014-09-14 12:26:36 -0700 (diff) (blame) 258         }
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 259     }
260
Tor Norbye ff37cc8 2018-03-09 07:56:23 -0800 (diff) (blame) 261     context.report(ISSUE, method, acquire, context.getLocation(release), message);
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 262 } catch (AnalyzerException e) {
263     context.log(e, null);
264 }
265 }
266
267 private static final int SEEN_TARGET = 1;
268 private static final int SEEN_BRANCH = 2;
269 private static final int SEEN_EXCEPTION = 4;
270 private static final int SEEN_RETURN = 8;
271
272 /** TODO RENAME */
273 private static class MyGraph extends ControlFlowGraph {
274     @Override
275     protected void add(@NonNull AbstractInsnNode from, @NonNull AbstractInsnNode to) {
276         if (from.getOpcode() == Opcodes.IFNULL) {
277             JumpInsnNode jump = (JumpInsnNode) from;
278             if (jump.label == to) {
279                 // Skip jump targets on null if it's surrounding the release call
280                 //
281                 // if (lock != null) {
282                 //     lock.release();
283                 // }
284                 //
285                 // The above shouldn't be considered a scenario where release() may not
286                 // be called.
287                 AbstractInsnNode next = Lint.getNextInstruction(from);
Tor Norbye 70257e6 2018-04-18 17:26:30 -0700 (diff) (blame) 288                 if (next != null && next.getType() == AbstractInsnNode.VAR_INSN) {
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 289                     next = Lint.getNextInstruction(next);
Tor Norbye 70257e6 2018-04-18 17:26:30 -0700 (diff) (blame) 290                     if (next != null && next.getType() == AbstractInsnNode.METHOD_INSN) {
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 291                         MethodInsnNode method = (MethodInsnNode) next;
Tor Norbye ff37cc8 2018-03-09 07:56:23 -0800 (diff) (blame) 292                         if (method.name.equals(RELEASE_METHOD)
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 293                             && method.owner.equals(WAKELOCK_OWNER)) {
294                             // This isn't entirely correct; this will also trigger
295                             // for "if (lock == null) { lock.release(); }" but that's
296                             // not likely (and caught by other null checking in tools)
297                             return;
298                         }
299                     }
300                 }
301             }
302         } else if (from.getOpcode() == Opcodes.IFEQ) {
303             JumpInsnNode jump = (JumpInsnNode) from;
304             if (jump.label == to) {
Tor Norbye 70257e6 2018-04-18 17:26:30 -0700 (diff) (blame) 305                 AbstractInsnNode prev = Lint.getPrevInstruction(from);
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 306                 if (prev != null && prev.getType() == AbstractInsnNode.METHOD_INSN) {
307                     MethodInsnNode method = (MethodInsnNode) prev;
Tor Norbye ff37cc8 2018-03-09 07:56:23 -0800 (diff) (blame) 308                     if (method.name.equals(IS_HELD_METHOD)
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 309                         && method.owner.equals(WAKELOCK_OWNER)) {
310                         AbstractInsnNode next = Lint.getNextInstruction(from);
Tor Norbye 70257e6 2018-04-18 17:26:30 -0700 (diff) (blame) 311                         if (next != null) {
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 312                             super.add(from, next);
313                             return;
314                         }
315                     }
316                 }
317             }
318         }
319     }
320     super.add(from, to);
321 }
322 }
323
Tor Norbye ff37cc8 2018-03-09 07:56:23 -0800 (diff) (blame) 324 /**
325  * Search from the given node towards the target; return false if we reach an exit point such as
326  * a return or a call on the way there that is not within a try/catch clause.
327  */
328 * @param node the current node
329 * @return true if the target was reached XXX RETURN VALUES ARE WRONG AS OF RIGHT NOW
330 */
Tor Norbye ff37cc8 2018-03-09 07:56:23 -0800 (diff) (blame) 331 protected static int dfs(ControlFlowGraph.Node node) {
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 332     AbstractInsnNode instruction = node.instruction;
333     if (instruction.getType() == AbstractInsnNode.JUMP_INSN) {
334         int opcode = instruction.getOpcode();
335         if (opcode == Opcodes.RETURN
Tor Norbye ff37cc8 2018-03-09 07:56:23 -0800 (diff) (blame) 336             || opcode == Opcodes.ARETURN
337             || opcode == Opcodes.LRETURN
338             || opcode == Opcodes.IRETURN
339             || opcode == Opcodes.DRETURN
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 340             || opcode == Opcodes.FRETURN
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 341             || opcode == Opcodes.ATHROW) {
342         return SEEN_RETURN;
343     }
344 }
345
Tor Norbye 2a8fd86 2019-07-01 14:05:18 -0700 (diff) (blame) 346 // There are no cycles, so no *NEED* for this, though it does avoid
347 // researching shared labels. However, it makes debugging harder (no re-entry)
348 // so this is only done when debugging is off
349 if (node.visit != 0) {
350     return 0;
351 }
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 352 node.visit = 1;
Tor Norbye 2a8fd86 2019-07-01 14:05:18 -0700 (diff) (blame) 353
Tor Norbye 940b0f9 2012-12-12 15:52:52 -0800 (diff) (blame) 354 // Look for the target. This is any method call node which is a release on the
355 // lock (later also check it's the same instance, though that's harder).
356 // This is because finally blocks tend to be inlined so from a single try/catch/finally
```

				357	// with a release() in the finally, the bytecode can contain multiple repeated
				358	// (inlined) release() calls.
				359	if (instruction.getType() == AbstractInsnNode.METHOD_INSN) {
				360	MethodInsnNode method = (MethodInsnNode) instruction;
				361	if (method.name.equals(RELEASE_METHOD) && method.owner.equals(WAKELOCK_OWNER)) {
				362	return SEEN_TARGET;
				363	} else if (method.name.equals(Acquire_METHOD) && method.owner.equals(WAKELOCK_OWNER)) {
				364	// OK
				365	} else if (method.name.equals(IS_HELD_METHOD) && method.owner.equals(WAKELOCK_OWNER)) {
				366	// OK
				367	} else {
				368	// Some non acquire/release method call: if this is not associated with a
				369	// try-catch block, it would mean the exception would exit the method,
				370	// which would be an error
Tor Norbye	2a8fd86	2019-07-01 14:05:18 -0700	(diff) (blame)	371	if (node.exceptions.isEmpty()) {
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	(diff) (blame)	372	// Look up the corresponding frame, if any
				373	AbstractInsnNode curr = method.getPrevious();
				374	boolean foundFrame = false;
				375	while (curr != null) {
				376	if (curr.getType() == AbstractInsnNode.FRAME) {
				377	foundFrame = true;
				378	break;
				379	curr = curr.getPrevious();
				380	}
				381	}
				382	
				383	if (!foundFrame) {
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	(diff) (blame)	384	return SEEN_RETURN;
				385	}
				386	}
				387	}
				388	}
				389	
				390	// if (node.instruction is a call, and the call is not caught by
				391	// a try/catch block (provided the release is not inside the try/catch block)
				392	// then return false
				393	int status = 0;
				394	
				395	boolean implicitReturn = true;
				396	List<Node> successors = node.successors;
				397	List<Node> exceptions = node.exceptions;
Tor Norbye	2a8fd86	2019-07-01 14:05:18 -0700	(diff) (blame)	398	if (!exceptions.isEmpty()) {
				399	implicitReturn = false;
				400	}
				401	for (Node successor : exceptions) {
				402	status = dfs(successor) status;
				403	if ((status & SEEN_RETURN) != 0) {
				404	return status;
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	(diff) (blame)	405	}
				406	}
Tor Norbye	2a8fd86	2019-07-01 14:05:18 -0700	(diff) (blame)	407	
				408	if (status != 0) {
				409	status = SEEN_EXCEPTION;
				410	}
				411	
				412	if (!successors.isEmpty()) {
				413	implicitReturn = false;
				414	if (successors.size() > 1) {
				415	status = SEEN_BRANCH;
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	(diff) (blame)	416	}
Tor Norbye	2a8fd86	2019-07-01 14:05:18 -0700	(diff) (blame)	417	}
				418	for (Node successor : successors) {
				419	status = dfs(successor) status;
				420	if ((status & SEEN_RETURN) != 0) {
				421	return status;
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	(diff) (blame)	422	}
				423	}
				424	
				425	if (implicitReturn) {
				426	status = SEEN_RETURN;
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	(diff) (blame)	427	}
				428	
				429	return status;
				430	}
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	(diff) (blame)	431	
				432	// Check for the non-timeout version of wakelock acquire
				433	
				434	@Nullable
				435	@Override
				436	public List<String> getApplicableMethodNames() {
				437	return Collections.singletonList("acquire");
				438	}
				439	
				440	@Override
Tor Norbye	2102127	2018-08-21 17:52:40 -0700	(diff) (blame)	441	public void visitMethodCall(
Tor Norbye	ff37cc0	2018-03-09 07:56:23 -0800	(diff) (blame)	442	@NonNull JavaContext context,
				443	@NonNull UCallExpression call,
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	(diff) (blame)	444	@NonNull PsiMethod method) {
Tor Norbye	0dedf90	2016-12-20 10:19:58 +0000	(diff) (blame)	445	if (call.getValueArgumentCount() > 0) {
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	(diff) (blame)	446	return;
				447	}
				448	
Tor Norbye	ff37cc0	2018-03-09 07:56:23 -0800	(diff) (blame)	449	if (!context.getEvaluator().isMemberInClass(method, "android.os.PowerManager.WakeLock")) {
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	(diff) (blame)	450	return;
				451	}
				452	
Tor Norbye	0dedf90	2016-12-20 10:19:58 +0000	(diff) (blame)	453	Location location = context.getLocation(call);
Tor Norbye	ff37cc0	2018-03-09 07:56:23 -0800	(diff) (blame)	454	LintFix fix =
				455	fix().name("Set timeout to 10 minutes")
				456	.replace()
Tor Norbye	d7439f1	2021-07-14 16:28:28 -0700	(diff) (blame)	457	.pattern("acquire\\s+\\(\\(\\s+\\)")
Tor Norbye	ff37cc0	2018-03-09 07:56:23 -0800	(diff) (blame)	458	.with("10*60*1000L /*10 minutes*/")
				459	.build();
Tor Norbye	a44cd04	2017-04-14 21:19:23 -0700	(diff) (blame)	460	
Tor Norbye	ff37cc0	2018-03-09 07:56:23 -0800	(diff) (blame)	461	context.report(
				462	TIMEOUT,
				463	call,
				464	location,
				465	""
				466	+ "Provide a timeout when requesting a wakelock with "
				467	+ "'PowerManager.WakeLock.acquire(long timeout)'. This will ensure the OS will "
				468	+ "cleanup any wakelocks that last longer than you intend, and will save your "
				469	+ "user's battery.",
				470	fix());
Tor Norbye	140957b	2017-01-23 07:08:07 -0800	(diff) (blame)	471	}
Tor Norbye	940b0f9	2012-12-12 15:52:52 -0800	(diff) (blame)	472	}